

Using Scaled Arithmetic

What do you do when your application's data doesn't map to the 8- and 16-bit integers offered by your project's microcontroller? You can use floating-pointing calculations or you have the option of scaling your numbers at a small cost in added software complexity.

Without using floating-point arithmetic, how would you add 12.34 and 987.654, multiply 3.1416 by 5.4, or divide 0.00456 by 98.7? This article shows how to perform basic arithmetic operations on fractional numbers using only integers.

These capabilities are important, because most low-end microprocessors do not have hardware-assisted floating-point math support. Microprocessor manufacturers seem to feel that floating-point math is a low priority in embedded systems. Fortunately, ANSI C compilers allow you to use floating-point math. There is, however, a cost: Floating-point libraries require extra ROM and RAM but more importantly, they require more processing time than integer math does. For example, floating-point addition could take hundreds of microseconds on a low-end 8-bit microprocessor, whereas it typically takes only a few microseconds to perform a 16-bit addition. Multiplication and division are often worse. As embedded system programmers, we are often confronted with the task of writing the fastest and smallest code possible for real-time operations. When we have to perform arithmetic

operations on fractional numbers, we generally turn to integer and fixed-point arithmetic.

In this article, I will be presenting code examples in C and performing operations on 16-bit integers. The concepts, however, apply to any integer size. You should keep in mind that implementations using assembly language would be much more efficient.

FIXED-POINT NUMBERS

An integer (either signed or unsigned) can be used to represent fractional numbers by moving the radix point within an integer, as shown in Figure 1. This is known as fixed-point. The first bit to the right of the radix point represents 0.5, the next bit 0.25, and so on. An integer value of 16 in the formats shown in Figure 1 represents 0.5. You may also say that the number 0.5 has been scaled by 2^5 (multiplied by 32). Another way to look at it is to say that 0.5 is equal to $16 * 2^{-5}$.

The unsigned integer of Figure 1 can be used to represent numbers having a range of 0.0 to 2,047.96875, and the signed integer can represent a number between -1,024.0 and 1,023.96875 (assuming two's complement). Both signed and unsigned numbers have a

resolution of $1/32$ nd (0.03125). You can use fixed-point to represent distances, surfaces, volumes, temperatures, pressures, and so on. Depending on the application, you can fix the position of the radix point elsewhere to suit the range of numbers you have to handle.

Figure 2 shows how to represent temperatures from -459.67°F (0° Kelvin, or absolute zero) to 2,048°F by using an 11-bit integer and a 4-bit fraction. An integer value of 11,528 represents a temperature of 720.5°F ($11,528 * 2^{-4}$). Using this format, temperatures can be represented with a $1/16$ th°F resolution.

When your program performs arithmetic operations (add, subtract, multiply, or divide) on fixed-point numbers, it actually manipulates integers. This is because microprocessors do not provide mechanisms to represent fixed-point numbers. The implication is that the programmer must keep track of the position of the radix points. To represent fixed-point numbers, I will use the

When programs perform arithmetic operations on fixed-point numbers, they manipulate integers.

following notation:

`<mantissa>S<exponent>`

where S means the mantissa needs to be scaled by 2^{exponent} to determine the value of the fixed-point number. The exponent is sometimes called the scale factor and can be used interchange-

FIGURE 1
Representing the fractional numbers.

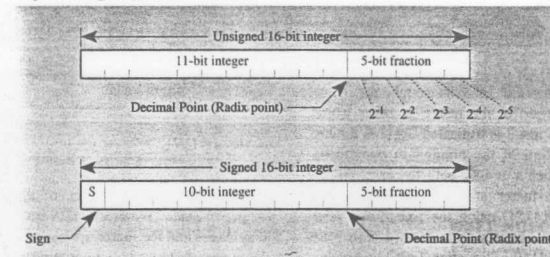
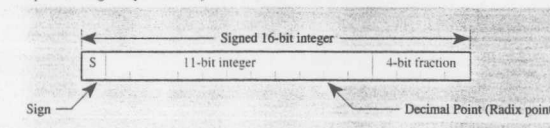


FIGURE 2
Representing temperatures from -459.67°F to 2047°F.



Using Scaled Arithmetic

ably. The mantissa is always an integer number. I use this notation to differentiate it from the floating-point notation $\langle \text{mantissa} \rangle \langle \text{exponent} \rangle$. The following examples show use of this notation:

55-3 represents $0.6250, 5 * 2^{-3}$, or $5/8$
 315-8 represents $0.1211, 31 * 2^{-8}$, or $31/256$
 -1235-16 represents $-0.001877, -123 * 2^{-16}$, or $-123/65,536$

The mantissa is shown in bold to emphasize that the fixed-point number is actually represented using an integer, whereas the exponent is maintained mentally (or on paper) by the programmer. Scaling is done to allow any number to be represented using a 16-bit (or more) integer. The position of the radix point is determined from the largest number you need to represent. Equation 1 shows how to obtain the mantissa and the exponent (scale factor) for any positive value x (unsigned value) between zero and 65,535.

$$\text{factor} = -\text{INT} \left(\frac{\log \left(\frac{65,535}{x} \right)}{\log(2)} \right)$$

$$\text{mantissa} = \text{INT}(2^{-\text{factor}} \times x + 0.5) \quad (1)$$

where INT() means we take the integer portion of the result. In other words, the result is truncated. log() is the logarithm of the number in parenthesis. When x is 0.0, both the mantissa and the factor are zero. To represent π (3.14159) using the fixed-point number notation, we would substitute 3.14159 in Equation 1 as follows:

$$-14 = -\text{INT} \left(\frac{\log \left(\frac{65,535}{3.14159} \right)}{\log(2)} \right)$$

$$51.472 = \text{INT}(2^{14} \times 3.14159 + 0.5)$$

Thus, 3.14159 is written as 51472S-14. (The actual value is 3.141602.) You

don't actually use Equation 1 in your code. Instead, use the equation to determine the integer equivalent of a floating-point number and its scale factor. Equation 2 shows how to obtain the mantissa and the exponent for a positive value x (unsigned number) greater than 65,535:

$$\text{factor} = \text{INT} \left(\frac{\log \left(\frac{x}{65,535} \right)}{\log(2)} \right) + 1$$

$$\text{mantissa} = \frac{x}{2^{\text{factor}}} \quad (2)$$

For example, the number 107,573 is represented as:

$$1 = \text{INT} \left(\frac{\log \left(\frac{107,573}{65,535} \right)}{\log(2)} \right) + 1$$

$$53786 = \frac{107,573}{2^1}$$

and written as 53786S1. In this case, we lose resolution, because we would need 17 bits to represent 107,573.

Equation 3 shows how to obtain the mantissa and the exponent for any signed value x between -32,767 and 32,767 inclusively, where:

$$\text{factor} = -\text{INT} \left(\frac{\log \left(\frac{32,767}{|x|} \right)}{\log(2)} \right)$$

$$\text{mantissa} = \text{INT}(2^{-\text{factor}} \times x + 0.5) \quad (3)$$

$|x|$ means the absolute value of the number to scale. When x is 0.0, both the mantissa and the factor are zero.

Equation 4 shows how to obtain the mantissa and the exponent for a signed integer less than -32,767 and greater than 32,767:

$$\text{factor} = \text{INT} \left(\frac{\log \left(\frac{|x|}{32,767} \right)}{\log(2)} \right) + 1$$

$$\text{mantissa} = \frac{x}{2^{\text{factor}}} \quad (4)$$

Natural Selection IAR C and Windows

8051
8096/196
68HC11
68HC16
Z80/180
64180
H8/500
H8/300
H8/300H
MELPS 7700
NEC 7800/78K
OKI 65K

IAR Systems' DOS and Windows based tools produce the same high quality, reliable code.

THE IAR TOOLS FOR EMBEDDED SYSTEMS DEVELOPMENT HAVE COME A LONG WAY. FROM ASSEMBLER TO C PROGRAMMING, AND FROM COMMAND-DRIVEN UTILITIES TO THE FULLY INTEGRATED IAR EMBEDDED WORKBENCH FOR WINDOWS.

IAR SYSTEMS' WORKBENCH OFFERS ALL THE PRODUCTIVITY BENEFITS OF WINDOWS, AND AT THE CORE IS IAR SYSTEMS' NEW OPTIMIZED COMPILER ENGINE.

EASE AND SPEED AROUND THE EDIT/COMPILE/DEBUG CYCLE ARE

THE MOST IMPORTANT FACTORS IN IMPROVING PRODUCTIVITY. WINDOWS ADDS NEW EFFICIENCY TO THE LEGENDARY QUALITY AND RELIABILITY OF IAR TOOLS, HELPING YOU WORK AND BRING PRODUCTS TO MARKET FASTER.

PLUG & PLAY INSTALLATION GETS YOU STARTED WITH THE WORKBENCH STRAIGHTAWAY, WITH NO CONFIGURATION PROBLEMS.

CALL NOW
FOR MORE
INFORMATION
AND A FREE DEMO
1-800-427-8868

IAR
SYSTEMS
advanced
EMBEDDED
SOLUTIONS

IAR SYSTEMS SOFTWARE, INC.
ONE MARITIME PLAZA
SAN FRANCISCO, CA 94111
TEL: 415-785-5500 FAX: 415-785-5503
IAR SYSTEMS LTD UK
TEL: +44 171 924 3334 FAX: +44 171 924 5341
IAR SYSTEMS AB SWEDEN
TEL: +46 18 16 78 00 FAX: +46 18 16 78 38
IAR SYSTEMS GMBH GERMANY
TEL: +49 89 470 8022 FAX: +49 89 470 9565

IAR C CODES COMPILERS WERE PREVIOUSLY LICENSED FOR DISTRIBUTION IN THE US AND CANADA UNDER THE ARCHIMEDES BRAND NAME

CIRCLE # 26 ON READER SERVICE CARD

Using Scaled Arithmetic

ADDING AND SUBTRACTING

To add or subtract two fixed-point numbers, the exponent of both numbers must be the same. For example, you could not add the signed fixed-point number 20480S-15 (0.6250) with 31745S-18 (0.1211) because they do not represent the same order of magnitude. In this situation, reduce the mantissa portion of the smaller of the two numbers (31745S-18) by dividing it by eight, so its scale factor is S-15. The number would be 3968S-15, or $3.968/12.768$. The result of the addition is 24448S-15, or 0.746094. Pretty simple, right? Actually, things get trickier when you add two numbers and the result exceeds unity. For example, when:

$0.99 + 0.99 = 1.98$

or:

To multiply fixed-point numbers, multiply the mantissa of the two numbers and add the exponents.

$32440S-15 + 32440S-15 = 64880S-15$

Here, the addition overflows, because the maximum value for a signed 16-bit fixed-point number can

only be 32,767! You can avoid the overflow by scaling both numbers to S-14 instead of S-15, as follows:

$0.99 + 0.99 = 1.98$

or:

$16220S-14 + 16220S-14 = 32440S-14$

You should be careful when you add or subtract two fixed-point numbers.

FIXED-POINT MULTIPLICATION

To multiply fixed-point numbers, multiply the mantissa of the two numbers and add the exponents. For example, we can multiply the two signed 16-bit fixed-point numbers:

$0.6250 * 0.1211 = 0.075688$

or:

QUICK! THINK OF A TOOL THAT GETS YOUR REAL-TIME PRODUCT TO MARKET FASTER

Having trouble? So were we. That's why we, as real-time developers, created the ObjecTime® toolset to reduce time-to-market by automating key development activities. With ObjecTime®, to rapidly develop complex event-driven software you can:

- Create reusable components and component frameworks
- Execute and validate graphical requirements and design models throughout the development cycle
- Automatically generate complete production-quality code from design models. These are not just code "headers" or skeletons, but include the hardest parts to get right—complex component behavior and component interconnection and interaction. Target operating systems include VRTX, pSOS+, VxWorks, HP-RT, HP-UX, AIX, and Solaris.

While other companies are debating the merits of academic methods and traditional CASE diagramming tools, your real-time product is shipping!



ISBN 0-471-59917-4

ObjecTime supports the ROOM method authored by Bran Selic, Garth Gullekson, and Paul Ward (co-developer of the Ward-Mellor method)



OBJEC TIME® ObjecTime Limited
1-800-567-TIME

Available on Sun, HP, and IBM RS/6000 workstations

Phar Lap's new TNT Embedded ToolSuite makes this an easy target.



The complete 32-bit embedded solution.

The 386 used to be a tough target for embedded systems. You had to hunt down tools from many vendors. And tool compatibility was hit-or-miss.

Now there's a simple one-source solution – the Phar Lap® 32-Bit TNT Embedded ToolSuite™.

Phar Lap tools make the widely-available 386 an easy 32-bit embedded target. Use a standard 32-bit Windows C/C++ compiler from Borland, Microsoft or MetaWare. Embedded development is easy because you use the same tools you trust for DOS and Windows development.

Best-of-breed tools.

Pick Microsoft CodeView or Borland Turbo Debugger – Phar Lap turns them into high performance embedded debuggers. Use your compiler's C/C++ run-time libraries unmodified in your embedded system. You get full ROMability and a built-in floating point emulator.

The TNT Embedded Kernel gets you into protected mode right away. You're up and writing code within hours, not months. And Phar Lap's LinkLoc, a one-step embedded linker, puts you in complete control. Locate code and data anywhere in memory.

The ROMINIT feature saves you from initializing RAM variables – it's all automatic. LinkLoc provides full symbolic information for C/C++ source level debugging with CodeView or Turbo Debugger.

Full 32-bit power.

Don't settle for 16-bit real mode when you can aim for full 32-bit performance and speed. Enjoy a flat memory model architecture. No more segments. And no more frustrating 1 MB memory limit.

**Phar Lap's
TNT Embedded
ToolSuite
has all
you'll need.**

- Support for All Industry Standard C/C++ Compilers: Microsoft, Borland, MetaWare
- 32-Bit Linker/Locator
- TNT Embedded Kernel
- CYBER, TDEAR Shells for Embedded Cross-Debugging
- Full Support for C/C++ Run-Time Libraries
- MS-DOS Compatible File System
- Floating Point Emulation Library
- Visual System Builder

On-target quality.

Phar Lap delivered the first 32-bit protected-mode toolkit for the 386 way back in 1986. Now more than 50,000 programmers use Phar Lap 32-bit products for both DOS and embedded applications.

So if your target is easier 32-bit development, Phar Lap is your source. Call to find out more.



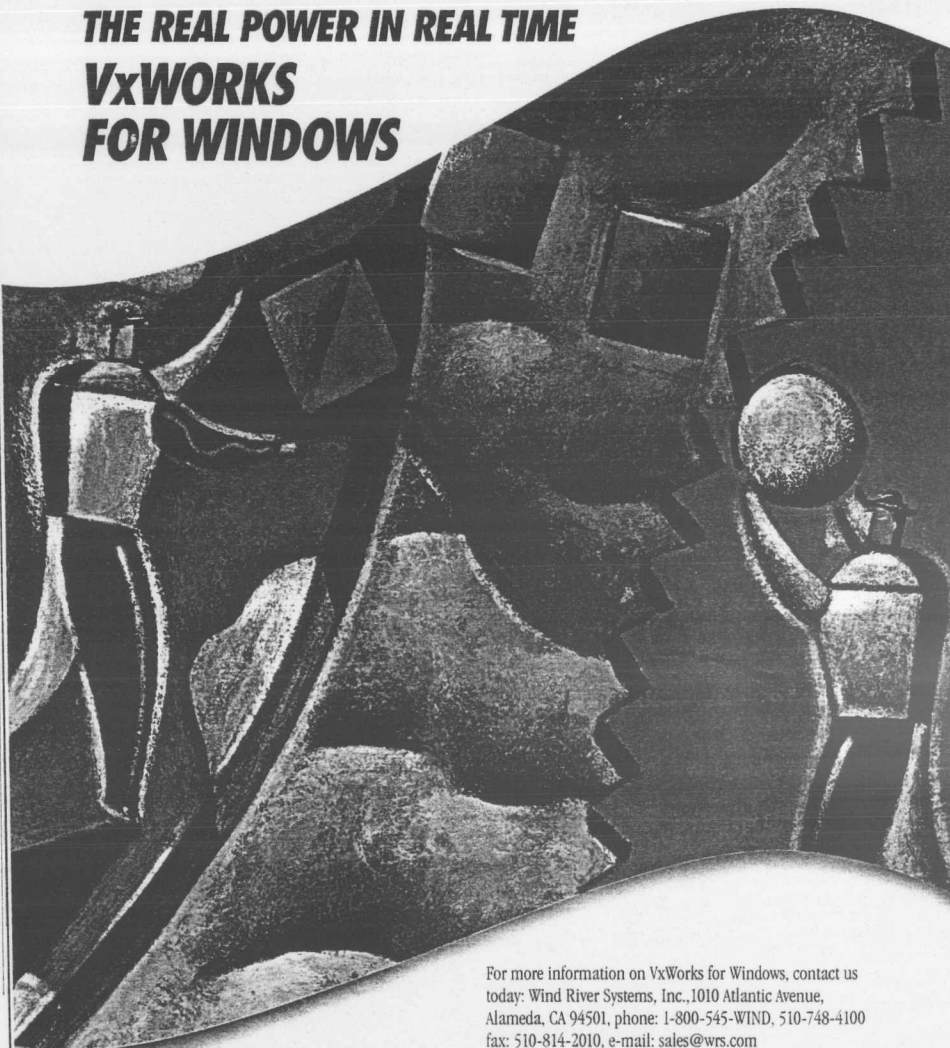
Phar Lap Software, Inc.

60 Aberdeen Avenue, Cambridge, MA 02138 617-661-1510 FAX: 617-876-2972

©1994, Phar Lap Software, Inc. All rights reserved. Phar Lap is a registered trademark, and TNT Embedded ToolSuite and ToolSuite are trademarks of Phar Lap Software, Inc. All other products and company names are trademarks or registered trademarks of their respective holders.

CIRCLE # 28 ON READER SERVICE CARD

THE REAL POWER IN REAL TIME VxWORKS FOR WINDOWS



For more information on VxWorks for Windows, contact us today: Wind River Systems, Inc., 1010 Atlantic Avenue, Alameda, CA 94501, phone: 1-800-545-WIND, 510-748-4100 fax: 510-814-2010, e-mail: sales@wrs.com

European Sales Offices:

France, 33-1-69-07-78-78; Germany, 49-89-96-09-49-44;
United Kingdom, 44-121-359-0981; Scandinavia, 46-8-707-3220



THE REAL POWER IN REAL TIME

© 1994 Wind River Systems, Inc. All rights reserved. VxWorks is a registered trademark of Wind River Systems, Inc.

CIRCLE # 30 ON READER SERVICE CARD

$$20480S-15 * 31745S-18 = 650137600S-33$$

When you multiply two signed 16-bit numbers, the result is a 30-bit number. Because of this, your C compiler needs to support signed longs (32-bit numbers). In the previous example, divide the number by 32768S-15 to obtain a signed 16-bit result. A division by 32768S-15 is a division by 1.0 and simply involves shifting the mantissa right 15 places. The result would be 19840S-18, or 0.075684.

For unsigned fixed-point numbers, multiplication yields a full 32-bit result. For example, $0.6250 * 0.1211$ looks like this:

$$40960S-16 * 63491S-19 = 2600591360S-35$$

A division of 65,536 would make the preceding result fit back into an unsigned 16-bit integer: 39681S-19, or 0.075686. The result is more accurate than its signed version, because more bits were used in the unsigned multiplication.

FIXED-POINT DIVISION

Division is always trickier and slower than multiplication is. For example, instead of dividing a number by 10, consider multiplying the number by 0.1 (or 26214S-18, signed). If you must perform a division, divide the mantissas and subtract the exponents:

$$\frac{0.2345}{-10.987} = -0.021343$$

or:

$$\frac{30736S-17}{-22501S-11} = -1S-6(-0.0115625)$$

The result is totally incorrect. The division produced a result of -1 and a remainder of 8,235. C compilers don't know what to do with remainders! To avoid this problem, scale the dividend by 32768S-15 (or 1.0) and remember that the final result has been multiplied by 32,768, as shown in the following:

$$(30736S-17 \times \frac{32768S-15}{22501S-11}) = -44760S-21 \text{ (or, } -0.021343)$$

The mantissa of the result doesn't fit in a 16-bit signed number. To prevent this situation, multiply by 16384S-14 instead of 32768S-15. The new operation would be:

$$(30736S-17 \times \frac{16384S-14}{-22501S-11}) = -22380S-20 \text{ (or } -0.021343)$$

An overflow will occur whenever the mantissa of the numerator is greater than the mantissa of the denominator. Your code will have to check for this situation.

FIXED-POINT COMPARISON

Comparing two fixed-point numbers presents a similar problem to adding and subtracting; the exponent of both numbers must be the same. For example, comparing 20480S-15 with 31745S-18 requires that you adjust the smaller of the two numbers to match the scale of the larger. 31745S-18 would become 3968S-15, or $3,968/32,768$. Once both numbers represent the same order of magnitude, comparing them simply requires that you compare the mantissas or the integer values.

Let's look at some examples of fixed-point arithmetic. Suppose you needed to compute the circumference of a circle that can vary in diameter from 1.22 inches to 20.8 inches. The circumference of a circle is given by:

$$\text{Circumference} = \pi * \text{Diameter}$$

Because diameters are positive quantities, we will use unsigned fixed-point numbers. π can be represented as 51472S-14 (actually 3,141602). Substituting the maximum diameter (20.8) into Equation 1, we obtain a scale factor of 11, so the fraction will require 11 bits. Because we are using an unsigned 16-bit integer, the mantissa portion will have five bits, as shown

BSO/TASKING 80C166



Harness the Power of the 166.

Optimized C compiler supports
165/166/167 family,
includes assembler, linker,
locator and all libraries

CrossView® Windows debugger
with Target ROM environment

Available for DOS and Unix

Small high performance RTOS

Evaluation boards for
all family members

On-line user manuals with
context sensitive help

EDE gives you access to all tools
through a common interface

Support hot line only a
toll-free call away

**BOSTON
SYSTEMS
OFFICE
TASKING**

**The hottest tools for
this fast microcontroller.**

Ask for your free demo disk and
our competitive benchmarks
1-800-458-8276

Fax **617-320-9212**

TASKING Europe +31 33 558584
TASKING Japan +81 334 050511

All trademarked names are the property of the respective owners

CIRCLE # 31 ON READER SERVICE CARD

the software equation is...



Simplify your application by using our Nucleus RTX or Nucleus PLUS real-time kernels. If your application needs TCP/IP or MS-DOS file support, let our Nucleus NET and Nucleus File products help.



Nucleus gives you a world of choices for prototyping. Our products run under MS-DOS, SUN-OS and SOLARIS 2. As a result you can build your application in friendly IBM-PC or SUN environments, using the Borland, Microsoft, or GNU development tools.



A near endless list of compiler/debugger tools are available to you. Our kernels are integrated with Microtec, Greenhills, Intermetrics, SDS, Sierra, Metaware, GNU, Intel, Microsoft, Borland and many others.



Find and kill the bugs in your application with the help of our multi-tasking debugging tools, which include Nucleus DBUG, Nucleus X-RAY and Nucleus X-RAY MTD.



The result is an application you can be proud of. Application development is a complicated process, but the equation is simplified by Nucleus.

Nucleus supports these processors
68xxx • 8086/186 • 80386/486 PM • DOS
Am29xxx • 1960 • IDT 305x • LR330xo • R4000
SPARCite • ARM • Hitachi H8/300H • Power PC
TMS320C3x/4x/2x/5x • DSP56xxx
68HC11/16 • M37702 • NEC 78k

for more information contact us at

(800) 468-6853



Accelerated Technology, Inc. Post Office Box 850245, Mobile, Alabama 36685 (205) 661-5770 Fax (205) 661-5788
11858 Bernardo Plaza Court, Suite 210-C, San Diego, California 92128

CIRCLE # 32 ON READER SERVICE CARD

in Figure 3.

The circumference of the circle is computed in C as follows:

```
WORD Circumference(WORD diameter)
{
    WORD x;

    x=(WORD)((51472L * (LONG)diameter)>>16);
    return (x);
}
```

Multiplying two 16-bit unsigned integers will yield a 32-bit result. This is why the multiplication is adjusted by dividing this result by 65,536 or shifting to the right 16 places. The exponent of the result is determined as follows. π is S-14 and multiplies the diameter S-11, resulting in an S-25 exponent. The right shift, however, is the same as dividing by 65536S-16, and the exponent of the result is S-9.

Our minimum circumference is obtained by substituting a 1.22 (2498S-11) inch diameter circle in the preceding code. The multiplication yields 128577056S-25. After the shift, the result is 1961S-9 (3.830078), which is within about 0.07% of the correct result, 3.832743. Our maximum circumference is obtained by substituting a 20.8-inch (42598S-11) diameter circle in the preceding code. The multiplication yields 2192604256S-25. After the shift, the result is 33456S-9 (65.343750), which is within 0.002% of the correct result, 65.345127. Fixed-point arithmetic produces large errors with small numbers (or numbers at the bottom of the scale) and decent results with large numbers. For large numbers, the improvement in accuracy is a direct result of using more bits.

Let's look at another example. We can use fixed-point arithmetic to con-

vert between degrees Centigrade (°C) and degrees Fahrenheit (°F). The conversion equation from °C to °F is shown as:

$$^{\circ}\text{C} = \frac{(^{\circ}\text{F} - 32) \times 5}{9} \quad (6)$$

To determine how to implement the conversion equation using fixed-point arithmetic, we need to know our range of temperatures. Suppose we are interested in temperatures from -50°F to 300°F. The range chosen forces us to use signed integer arithmetic. Also, we need nine bits to represent temperatures of up to 300°F, and six bits will represent fractional degrees. The bias of 32°F is represented as 2048S-6, and the constant multiplier $\frac{5}{9}$ can be represented as 18204S-15. The code to perform the conversion is:

```
WORD FtoC(WORD f)
/* f is scaled S-6 */
{
    return (((LONG)(f - 2048) * 18204) >>
        15);
    /* Result is S-6 */
}
```

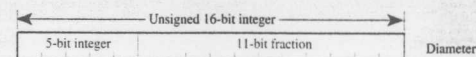
The temperature in °C is also scaled S-6 (or S-6 * S-15/S-15). In other words, the integer result is 64 times larger than the temperature it represents. Substituting 200°F (or 12800S-6 = 200 * 64) in the previous code yields a returned value of 5973S-6, or 93.328°C. (The actual value is 93.331.) WORD and LONG are typedefs, so they represent a signed 16-bit and a signed 32-bit integer, respectively.

Performing the conversion from °C to °F is just as simple. The equation is:

$$^{\circ}\text{F} = \frac{^{\circ}\text{C} \times 9}{5} + 32 \quad (7)$$

FIGURE 3

Fixed-point representation for circle diameter.



BSO/TASKING R3000



**No Risk.
Fast Applications.
No Competition.**

The most efficient software development solution available for R3000 applications

Optimizing ANSI C compiler, assembler and incremental linker

ANSI C, run-time and floating point libraries

Multi-tasking locator

CrossView® Target ROM debugger

EDE gives you access to all tools through a common interface

CrossView® and our Board Support Packages provide complete integrated solutions for all LSI LOGIC and IDT R3000 evaluation boards

**BOSTON
SYSTEMS
OFFICE
TASKING**

No argument, the best is BSO/TASKING.

Ask for your free demo disk and our competitive benchmarks
1-800-458-8276
Fax 617-320-9212

TASKING Europe +31 33 558584
TASKING Japan +81 334 050511

All trademarks names are the property of the respective owners

CIRCLE # 33 ON READER SERVICE CARD

Using Scaled Arithmetic

Again, the 32°F constant is 2048S-6, and the constant multiplier 9/5 is 29491S-14. The conversion code is:

```
WORD Ctof(WORD c)
/* c is scaled S-6 */
{
    WORD x;
    x = ((LONG)c * 29491) >> 14;
    return (x + 2048);
}
```

To obtain an S-6 result, divide the result of the multiplication by 16,384 instead of 32,768. Substituting 93°C (5952S-6=93*64) in the preceding code yields a returned value of 12761S-6, or 199.39°F. (The actual value is 199.40.)

In our third example, let's suppose the temperature used in the previous

example was obtained from a 10-bit analog-to-digital converter (ADC). The ADC produces a binary representation of the temperature. The temperature read by the sensor is given by Equation 8 (a temperature of -50°F produces zero ADC counts, 0°F is about 146 ADC counts, and 300°F is 1,023 ADC counts).

$$Temperature(^{\circ}F) = (ADC_{counts} - 146.143_{counts}) * 0.342131(^{\circ}F_{count}) (8)$$

The ADC can be scaled to fit into the upper portion of a signed 16-bit integer (or shifting left five places). To perform the subtraction, we need to scale 146.143 by the same value, or $146.143 * 32 = 4677S-5$. The gain (0.342131) obtained by using Equation 3 is 22421S-16. The temperature at the sensor is thus given by:

```
WORD RdTemp(WORD adc_counts)
/* Assumes 10-bit ADC */
{
    WORD cnts;
    WORD temp;
    cnts = (adc_counts << 5) - 4677;
    temp = (WORD)((LONG)cnts * (LONG)22421) >> 15L;
    return (temp);
} /* Result is scaled S-6 */
```

The result's scale factor is given by:

$$S-6 = S-5_{(ADC_{counts} << 5)} * S-16_{(0.342131)} / S-15_{(>> 15)}$$

The subtraction is done separately, because a good compiler should perform this operation using 16-bit arithmetic instead of 32-bit. (16-bit subtraction would be faster.) Counts and gain are then converted to LONG, because the multiplication needs 30-bit precision. The result is divided by 32,768 so it fits back into a 16-bit signed variable. Finally, the temperature is returned in °F scaled S-6. Obtain the temperature to the nearest degree by first adding 32 (0.5) and dividing the result by 64, which rounds the result.

Let Your Development Fly!

Choose Hitek professional tools for your embedded microprocessor design and get your project off the ground ahead of the schedule. Hitek builds all types of micro-processor development tools, from sophisticated in-circuit emulators to remote debuggers, monitors and simulators. Complete solutions are available for:

- the complete 8051 family
- 80C286, 80C186/188, 80486, 80387
- 80286, V20, V30, V40, V50
- 80386/286/386/486
- 80C15/16/187
- 80C11 family
- 80C01, 80C02, 80C03, 80C04

hitek

Call Hitek for your free demo package and learn how smooth and efficient development with professional tools really can be.

HITOOLS Inc.
2055 Gateway Place
Suite 400
San Jose, CA 95110
Tel: (408) 451-3000
Fax: (408) 441-0400

Please see us at the Embedded Systems Conference Booth #650

hitek

teletest 32

NEW! AX108
80C186/188 in-circuit emulator

CIRCLE # 34 ON READER SERVICE CARD

58 EMBEDDED SYSTEMS PROGRAMMING MARCH 1995

Man Acquires ESp, Begins Seeing Invisible Bugs.

Don't read the tabloids. Don't watch *A Current Affair*. This breaking story is brought to you by Integrated Systems. And there's nothing supernatural about it.

Our brand-new Embedded System profiler, ESp[®] for short, lets you debug embedded applications with your very own eyes.



One mouse click, and ESp graphically displays all of your application's activities. So you see what works, and what doesn't, without having to plow through line after line of code. You select the information and detail you want recorded. And define which events trigger the recording.

That way, you don't waste time with non-essential data. Whether you're debugging on the fly, or doing a postmortem analysis.

ESp also allows you to share target boards over a network. And lets you debug applications from remote sites by dialing in.

And because it uses just 22k of target memory, ESp can be used with truly embedded applications.

If you had ESp, you'd know all the facts. But you don't. So call 1-800-270-1115 for more info and a free demo.



CIRCLE # 35 ON READER SERVICE CARD

IEEE Real-Time Technology and Applications Symposium



Chicago, May 15-17, 1995



SPONSORED BY
IEEE COMPUTER SOCIETY T.C. ON REAL-TIME SYSTEMS
IN COOPERATION WITH THE OFFICE OF NAVAL RESEARCH

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation but also on the time at which the results are produced. Examples include C⁴I, embedded systems, process control, avionics, multimedia, and intelligent vehicle and highway systems. This symposium is a major forum for the exchange of emerging principles and practices underlying real-time technology and its applications. It will consist of tutorials, panels, and paper presentations on topics such as operating systems and scheduling, standards, management, testing, programming environments and tools, communication networks, architectures, performance modeling and measurement, case studies, and applications.

General Chair
Ted Baker

Program Chair
Wei Zhao

Treasurer
Ted Giering

Publicity Chair
Raj Rajkumar

Local Arrangements Co-Chairs:
Jeffrey Tsai, Chengwen Liu

Ex-Officio: (RTS-TC Chairs)
John Stankovic, Al Mok

ADVANCE PROGRAM

For advance program and other information, contact Dr. Ted Baker, Department of Computer Science, Florida State University, Tallahassee, FL 32306-4019, phone (904) 644-5452, fax (904) 644-0058, email baker@cs.fsu.edu

HOTEL RESERVATION

Contact the Bismarck Hotel, 171 West Randolph St. Chicago, IL 60601, phone: (312) 236-0123 or (800) 643-1500, fax: (312) 236-3177, mentioning RTAS'95, before April 23, 1995.

REGISTRATION FORM

Mail to: Linda Buss, RTAS'95 Registration,
Rt. 1 Box 187B, Menomonie, WI 54751,
phone (715) 235-0487, fax: (715) 232-6244,
email: rtas95@ada.cs.fsu.edu

Name: _____
Affiliation: _____
Address: _____
Phone: _____ Fax: _____
Email: _____
IEEE Membership No: _____

Fees:	Before Apr 15	After Apr 15
IEEE Members	\$310	\$390
Non-Members	\$390	\$490
Full-time Students	\$160	\$200

The student fee includes all the events of the symposium. To receive student rate, students are required to have advisor's name and signature at the time of registration.

Advisor name: _____
Signature: _____

Payment can be made by check, money order, or credit card. Please make checks or money orders payable, in US currency, to RTAS'95.

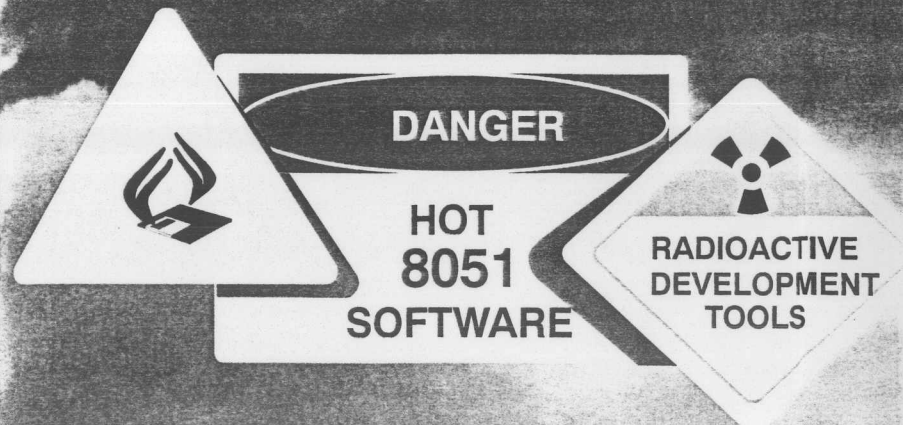
Credit Card:
☐ Visa ☐ Mastercard ☐ American Express
Credit Card Number: _____
Cardholder Name: _____
Credit Card Expiration Date: _____
Total Charges Authorized: _____
Signature: _____

I have demonstrated how to perform simple arithmetic operations using integers instead of floating-point. To use fixed-point arithmetic, however, you need to know the range of values the variables can take. Fixed-point arithmetic operations will generally execute much more quickly than their floating-point counterparts, because most microprocessors are good at performing integer operations. This performance is at the expense of accuracy and code complexity. The only way you can improve the accuracy is to use more bits. In a number of situations, however, 16-bit arithmetic is more than sufficient accuracy. Fixed-point works well when the dynamic range of the numbers used is small. **ES**

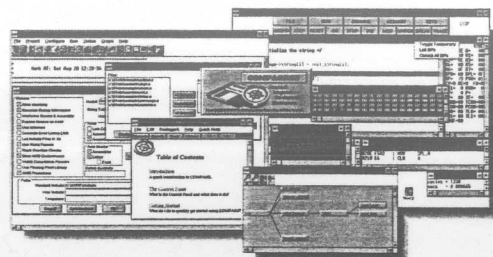
Jean J. Labrosse has an MS in electrical engineering from the University of Sherbrooke in Québec, Canada, and has been developing real-time software for over 10 years. Labrosse currently works at Dyalco Controls in Fort Lauderdale, FL. He wrote *µC/OS*, *The Real-Time Kernel*, describing the internals of a small real-time preemptive multitasking kernel, and *Embedded Systems Building Blocks: Complete and Ready-to-Use Modules in C*, from which this article is excerpted. (Both books available through R&D Publications, Lawrence, KS).

SUGGESTED READING

1. Crowell, Charles. *Floating-Point Arithmetic with the TMS32010*. Houston, TX: Texas Instruments Inc., 1986.
2. Knuth, Donald E. *The Art of Computer Programming, Vol. 2 Seminumerical Algorithms*. Reading, MA: Addison-Wesley, 1981.
3. Labrosse, Jean J. *Embedded Systems Building Blocks, Complete and Ready-to-Use Modules in C*. Lawrence, KS: R&D Publications, 1995.
4. Morgan, Don. *Numerical Methods, Real-Time and Embedded Systems Programming*. San Mateo, CA: M&T Publishing, 1992.



Sometimes you have to do something a little crazy to get noticed. PLC's COMPASS/51™ Integrated Development Environment for the 8051 family is about to get your attention. Now you can get *the first complete Windows™ hosted 8051 software development system* for just \$695. That's more than 60% off the regular \$1695 price!



You'll get an optimizing ANSI C Compiler that stacks up against the best of them, a fast Macro Assembler, a powerful Linker/Locator, a feature-packed Source Level Debugger, and a validated Instruction Simulator, all running under Windows 3.1! You'll also get complete on-line manuals for each tool. No more cumbersome books to clutter your desk!

You'll want to kiss your DOS window goodbye since you can design, code, and test your software without ever leaving Windows! Turn waiting time into productive time with COMPASS/51's advanced Program Visualization, Visual Debugging, and background processing capabilities.

These are NOT evaluation tools. There are no limitations, no missing features, just a solid set of quality development software at a great price!

Call today, and get your copy of PLC's COMPASS/51, the hottest new 8051 development system available.

\$695

This coupon entitles me to one copy of PLC's COMPASS/51™ IDE for Windows at ~~\$1695~~. I understand that this offer is good only through May 31, 1995.

Name: _____

Company: _____ State: _____ Zip: _____

Address: _____

Phone: _____

Form of Payment: ☐ Visa ☐ Master Card ☐ Check (Enclosed)

Card Number: _____ Exp: _____

International: (817) 599-8363 Production Languages Corporation
US Only: (800) 525-6289 P.O. Box 109
FAX: (817) 599-5089 Weatherford, TX 76086

CIRCLE # 37 ON READER SERVICE CARD

Windows is a trademark of Microsoft Corporation. COMPASS/51 is a trademark of Production Languages Corporation.

CIRCLE # 36 ON READER SERVICE CARD